

FEATURE ARTICLE

Tracey Lee
& Kok-Leong Ong

Speeding and Slimming Your Port Access

A Different Way of Reading from the PC Parallel Port

While many PC manufacturers today are supplying their machines with true bidirectional parallel ports, reading data in with older machines is still tricky. Here's one method that requires no changes to the PC.

When we came across the DDT-51 8051 emulator project presented by Steve

Ciarcia several years ago (BYTE 1988), we decided to improve on his implementation. The DDT-51 communicates with the PC using its parallel printer port. Steve modified a standard port for bidirectional operation, but we wanted to use an unmodified port. We decided to forego the soldering iron in favor of a software solution.

We chose to read data a nybble at a time, using one of the other parallel port pins to indicate which of the two nybbles transferred. Of course, the software later combines both nybbles. This method slows down data transfer, especially if the software is written using a high-level language, and uses one extra pin of the port interface. What is more important, however, is that the port and device have to make that one extra connection. This means that cable

and connectors have to be one bit wider.

To accomplish this variation, software handling the transfer needs to generate a READ signal using one of the parallel port pins. We explored the possibility of using this signal to indicate the order of nybble read. Assuming a negative logic READ, our solution was to use the low level (active) to indicate one nybble and to use the high level (inactive) to latch the data and gate the next nybble to the interface.

Thus, we save the two instructions needed to manipulate the nybble indicator bit in Steve's approach. We also save on one extra connection and do not have to perform surgery on the parallel card as suggested by Steve.

By the way, although we use the DDT-51 as our example, we understand that it has been discontinued as a product. The techniques described in this article, however, can be applied to parallel ports in general or with other devices that interface to the PC in a similar manner.

THE PARALLEL INTERFACE

Before getting into the details of the solution, let's take a look at the PC parallel-port interface. Enhancements to the interface starting from the PS/2

PC Address	Bit No.	Corresponding Pin Name	Pin No. DB-25	I/O Direction
Base	0	D0	2	Output Only
	1	D1	3	
	2	D2	4	
	3	D3	5	
	4	D4	6	
	5	D5	7	
	6	D6	8	
Base + 1	7	D7	9	Input Only
	0			
	1	Not Used	NA	
	2			
	3	-Error	15	
	4	SLCTD	13	
	5	PE	12	
Base + 2	6	-ACK	10	Input and Output
	7	Busy	11	
	0	Sirobe	1	
	1	Auto FD XT	14	
	2	-Init	16	
	3	SLCT IN	17	
	4	IRQ Enable		
	5			NA
	6	Not Used	NA	
	7			

Table 1—The functions on original PC parallel port interface are accessed through three consecutive I/O ports.

Listing 1—This sample code reads in a byte of data using the NYB_IND technique. If pin 16 is used as NYB_IND, then pin 17 becomes the RD pin and both pins are negative logic.

```
Port [$37A] := $00;           (Read first nybble)
FirstNibble := (Port [$379] And $F0) SHR 4;

Port [$37A] := $08;           (Get next nybble)

SecondNibble := Port [$379] And $F0;
Port [$37A] := $0C;           (Pull RD high)

ByteData := FirstNibble Or SecondNibble; (Combine the nybbles)
```

series of computers have enabled bidirectional 8-bit data transfer. Recently, a DMA (direct memory access) function was included as well!

However, we will restrict ourselves to the "original" parallel port, which consists of a 25-pin D connector (for pin assignments, see Table 1). Each port has three addresses associated with it in the I/O map of the PC. The first parallel port, or LPT1, has a base address of 378H, so its addresses would be 378H, 379H, and 37AH. Each address allows us to read or write certain data to the parallel port to control the logic state of each pin.

Due to the design of the standard parallel port, certain pins have negative logic while others use positive logic. Port 379H involves inversion of certain bits while port 37AH does not when used for input. As noted, we use port 378H to output 8 bits, port 379H to input up to 5 bits, and port 37AH to input or output up to 5 bits.

For faster access and greater flexibility, software should directly access these parallel ports through the three addresses associated with it. As used in the PC, parallel port 1 has address 378H for writing a byte of data to output to the parallel port pins, address 37AH enables writing control signals, and address 379H gives input signals from a device. Therefore, to output a byte of data, the program writes a character to address 378H, and input may be done at address 379H and 37AH.

THE NYBBLE INDICATOR APPROACH

Assume for a moment that we are exchanging data between two computers using the parallel port. The receiving computer needs to read data from the parallel port, but can only do so five bits at a time. This is an inconvenient quantity, so we will just use four of these bits for a nybble of data.

Typically, when you manipulate nybbles with software, you designate one pin of the port to act as a nybble indicator. The sending program is responsible for splitting the data into two nybbles and transfers it via four of the five input pins on the parallel interface.

One example of such a software transfer may be seen in the code segment in Listing 1. If the sending party is a piece of software, then the receiving program indicates with the NYB_IND pin which nybble is to be sent.

IN HARDWARE

If instead hardware is used to transmit data, a quad 2-to-1 multiplexer solves the problem. Figure 1 shows the schematic connecting the device to a standard parallel interface.

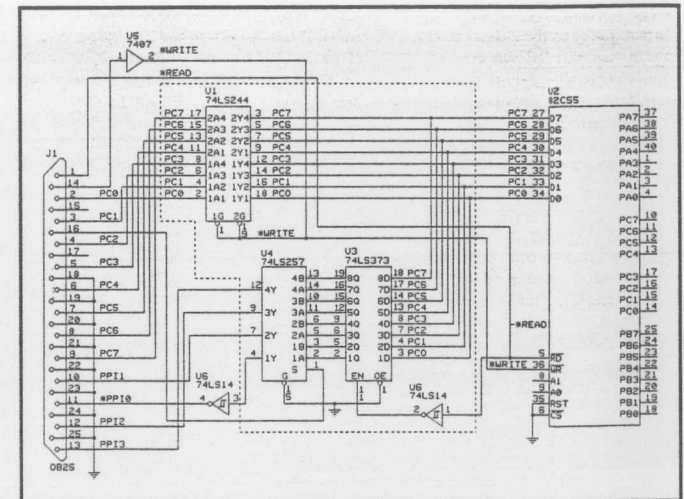


Figure 1—In the Nybble Indicator approach, a bit is used to designate whether the high or the low nybble is being transferred.

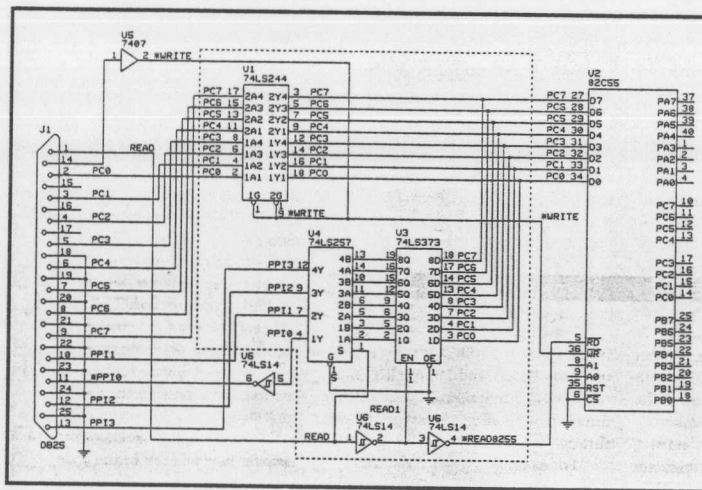


Figure 2—The Read Edge approach differs from the previous method in its use of the RD line.

In our example, the nybble indicator uses pin 16, which corresponds to bit 2 on port address 37AH.

Looking at the original DDT-51, the data bus consists of pins 2-9 on the parallel-port connector—it became bidirectional through hardware modifications. If the port is left unmodified, we have to add a 74LS244 (U2), which is used to isolate the data bus of the 8255 from the parallel port. U2 is required as the D0-D7 pins on the standard parallel port are constantly outputting data to the data bus, which causes problems during an input operation. Pins 11, 10, 12, and 13 are now used for input to the PC, which correspond to bits 7, 6, 5, and 4 on port address 379H.

The dotted area in Figure 1 shows the change to the original circuit. In this approach, the sequence of reading a byte begins by setting the READ line high. To initiate a read operation, the line changes to low. The software then pulls the NYB_IND (active) high and reads the first nybble from the 74LS257. It then saves the nybble and pulls the NYB_IND low to read in the next nybble. The read operation ends by setting the NYB_IND and READ to high again. The -A/B pin of the 74LS257 is connected to the NYB_IND pin on the parallel port.

THE READ EDGE APPROACH

Now, let's take a look at what we did. For reasons that will be apparent later, we will call this the *Read Edge method*. Figure 2 shows the schematic with the new circuitry. We use a pair of Schmitt-triggered inverters (U7, U8) to clean up the READ line. The difference between the methods hinges primarily in the use of the READ signal. Here, together with the additional ICs, as shown in the figure, it is also used as the nybble indicator.

When the software needs to read a byte of data, it pulls the READ line

lower four bits of the byte and outputs this nybble. The READ signal, which acts like the NYB_IND, goes to the -A/B input of the multiplexer to determine which nybble to place on the input pins. In this case, it is low for the lower nybble and high for the higher nybble. These nybbles are stored in the PC.

The software then pulls the READ line high. The LE on U4 then goes low as the first Schmitt-triggered inverter (U7) reverses the logic level of the signal. The falling edge of this signal latches the output from the 8255.

Listing 2—The READ8255 routine in the DDT-51 software has to be modified to work with the Read Edge hardware.

```
Function 8255 : Byte;
Var HighByte, LowByte : Byte;
Begin
    Port [CTLPort] := CstbCR;           (DDT-51 port values)
    Port [CTLPort] := Cread;

    Port [CTLPort] := CstbRD;           (Pull RD low)
    LowByte := (Port [InpPort] And $F0) SHR 4;

    Port [CTLPort] := Cnull;           (Pull RD high)
    HighByte := Port [InpPort] And $F0;

    Read8255 := HighByte Or LowByte;    (Combine nybbles)
End;
```

low. This causes U5, a quad 2-to-1 74LS257, to output the low byte to the input portion of the parallel port. (Again, we are using bits 7, 6, 5, 4 of port 379H, which correspond to pins 11, 10, 12, 13 on the parallel port.) The signal causes U4, a 74LS373 latch, to pass data through to U5.

On receiving the READ8255 signal, the 8255 places the data on the bus, which connects to U4. Since U4 is a transparent latch, data passes through it when LE, which is connected to the READ1 signal, is high. This allows the data to appear at the input of U5, which selects the

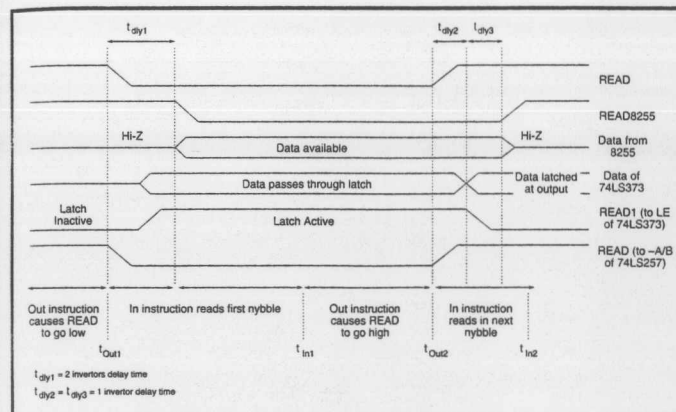


Figure 3—A timing diagram helps in analyzing the Read Edge approach for any possible delay problems.

After the rising edge of the READ8255 signal, the 8255 releases the data bus to a high-impedance state at the same time or before U4 is able to latch the data. The second inverter (U8) delays the READ8255 signal so that U4 (the 74LS373) has time to latch the data.

Latching is done before the READ1 signal propagates through the second 74LS14, which pulls the READ8255 line at the 8255 high. When this happens, the output from the 8255 floats. Looking now at U5, the 74LS257 multiplexer, the high READ signal outputs the high byte of the data. The software reads the nybble and then combines them to form the final byte. Figure 3 shows the timing diagram for the read operation.

TIMING ANALYSIS

Let us analyze the timing of the signals to see the potential and limits of this approach. Look at time t_{out1} in Figure 3. After the CPU brings the READ line low, an IN instruction reads in the first nybble. Because of the timing delay of t_{dly1} , the first nybble must be ready to be read by t_{in1} . However, the read for the second nybble always finds data ready at t_{in2} .

Now, if the CPU wants to do a write to the 8255 instead, the time t_{dly2} and t_{dly3} causes the 8255 to change from a READ state that is much slower.

The inverter delays are on the order of 10 ns using standard LS logic. An IN instruction takes hundreds of nanoseconds to execute for a PC. Some of you may frown at using delays to achieve our objectives, but we have demonstrated that it does work. Incorporating the circuitry into a PLD may lessen some of the layout problems.

THE SOFTWARE

So far, we have been talking about the hardware that converts bytes into nybbles for the PC parallel port. However, the approach of reading such a byte of data also requires the use of software to generate the necessary read signal and assemble the two nybbles into a byte.

In the DDT-51 project, we modified the READ8255 function to achieve this. Listing 2 shows the code samples written in Turbo Pascal. Note that there are two variables HighByte and LowByte, both of the data type BYTE.

When a statement calls the function READ8255, the READ line on the parallel port goes active (low). This causes the hardware to respond with the lower nybble at the input pins. The software then reads the nybble from the input pin through the address 379H.

Since each read operation at address 379H yields a byte of data, the software ignores any unused bits in the byte of data and zeros them before storing them into the variables. From Table 1, we see that the four input pins lie in the upper four bits of the byte.

Depending on the state of the READ signal, the software either shifts the nybble four bits to the right or no shifting is performed at all.

As you can see in Listing 2, the first nybble read represents the low

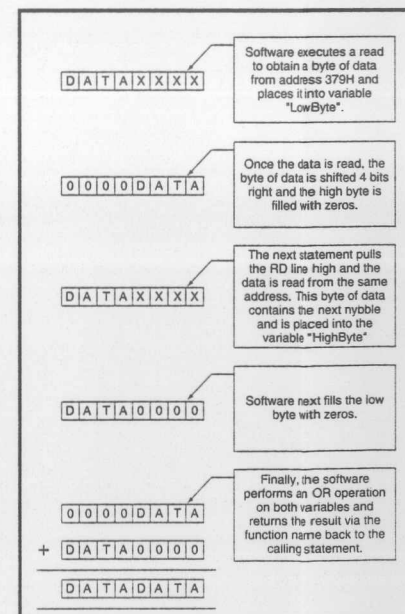


Figure 4—The nybble manipulations needed to recover a byte of data when only four bits of the data are transferred at a time are readily coded in most languages.

INTRODUCING THE SMARTEST GAMES IN TOWN

Micromint ups the ante with DOMINO and BLACKJACK

The Domino-52 microcontroller is a "supercomputer" in less than ¼ cubic inches. Domino is a plug-and-go module, just attach +5 V and a terminal or network. A simple keyed sequence saves it as an autostarting program in nonvolatile memory. We've packed the most essential elements of a microcontroller into one tiny package, making Domino-52 a winning combination. Just look at what you get:

DOMINO
Microcontroller



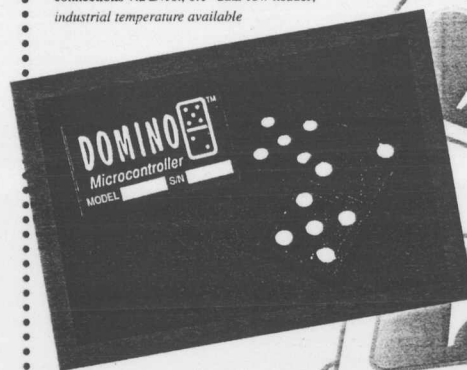
starting at
\$79.00

starting at
\$139.00

BLACKJACK
TELECONTROLLER



- Processor: 80C52 with ROM-resident, full floating-point BASIC; instant "enter and save" command programming; 11.0592-MHz clock
- Memory: 32K bytes SRAM and 32K bytes EEPROM
- Serial I/O (up to 19200 bps): Full-duplex RS-422 and RS-232A, half-duplex RS-485, I²C bus
- Parallel I/O: 12 bits available, 3 bits shared with ADC and I²C
- A/D converter (optional): 2-channel, 12-bit, 10k samples/s
- Interrupts and timers: Two interrupts and three timers; interrupt lines can be PWM outputs, timer inputs can directly read frequency and period
- Power, size, and connection: +5V @ 30 mA; 1.75"x1.062"x0.4" potted; connections via 2x10, 0.1" dual-row header; industrial temperature available



- Processor: 80C552 with firmware monitor, 14.7456-MHz clock, two interrupts, and three timers, watchdog timer
- Memory: 32K bytes SRAM, 32K bytes ROM, and 128K bytes EEPROM (paged)
- Serial I/O (up to 38400 bps): Full-duplex serial TTL, I²C bus
- Parallel I/O: 20 bits available, 4 bits shared with RTC, two PWM outputs
- Hardware real-time clock
- A/D converter: 8-channel, 10-bit, 20k samples/s
- Modem (optional): 2400-bps data modem (2400-bps data/fax modem optional)
- Power, size, and connection: +5V @ 55 mA, @ 95 mA (w/modem active) 2.75"x1.375"x.5" potted; 1.2" dual-row 48-pin DIP header; industrial temperature available

- Domino-52 **\$79.00**
- Domino-52 w/12-bit ADC **\$99.00**
- BASIC-52 programmer's manual, 128 p. **\$15.00**
- Domino PC utilities and development **\$99.00**
- Domino screw-terminal proto-connecter **\$19.00**

- BlackJack-552 (w/o modem) **\$139.00**
- BlackJack-552 with 2400-bps modem **\$199.00**
- BlackJack demo board (BlackJack module extra) **\$99.00**
- BlackJack utilities and development package **\$99.00**



MICROMINT, INC. • 4 Park Street • Vernon, CT 06066 • (203) 871-6170 • Fax (203) 872-2204
in Europe: (44) 0285-658122 • in Canada: (514) 336-9426 • in Australia: (3) 467-7194 • Distributor Inquiries Welcome!

Domino-52 and BlackJack-552 logos and trademarks are mutually owned by Xecom, Inc. and Micromint, Inc.

byte. The software shifts this nybble four bits to the right and fills the upper nybble with zeros. It then pulls the READ line high to read the next nybble. The second nybble, which represents the high byte, need not be shifted at all since it is already in its correct position. The lower nybble is then filled with zeros.

Once the software stores the two nybbles in the variables HighByte and LowByte, they are ORED together. The last statement in the function then places the result into the function name, which returns the value to the calling statement. Figure 4 illustrates the process of the software performing

the read and assembling the two nybbles.

ASSEMBLY LANGUAGE

Writing the code using a high-level language to read the nybbles and assembling it to form the final byte of data may be too slow for certain applications. To solve this, you can use an optimizing compiler to produce a faster routine or simply write the READ8255 function in assembly language to begin with.

We have converted the READ8255 function to C and used Borland C++ 3.1 to compile the program. By turning on the speed-optimization option, the

compiler was able to generate a fast routine for the read operation that was almost as fast as one we hand-optimized in assembler.

Turbo Pascal, however, produces a final assembly listing with some slight overhead in the code. The overhead may be avoided by simply writing the assembly version. We provide a sample of the assembly version of our READ8255 in Listing 3.

CONCLUSIONS

Although the Read Edge approach allows the software to read a byte of data using a standard parallel port with a single READ signal, it cannot be used on two PCs without this hardware between them.

But, this method is one of the many ways to read data using a parallel port. Its main advantage lies in the use of the READ signal to differentiate the nybbles in the data byte. This saves one pin on the parallel port which may be used for other purposes such as a control line for the attached device. Furthermore, this technique reduces the complexity of the software as less coding is required. □

Tracey Lee specializes in microsystems in the Department of Electronics, Computer, and Communications Engineering at the Singapore Polytechnic. He worked for IBM Singapore for 8 years and has a B.Eng. from Singapore University and a M.Eng. from Nanyang Technology University. He may be reached at spec02@solomon.technet.sg.

Kok-Leong Ong recently finished his studies at the Singapore Polytechnic where, together with two other students, he worked on a final year project entitled "8051 Emulator."

REFERENCE

Garcia, Steve. "Why Microcontrollers?" BYTE, Aug.-Sept., 1988.

I R S

410 Very Useful
411 Moderately Useful
412 Not Useful

Listing 3—The routines to read and write data to the standard PC parallel port are generic enough to be useful for most applications. However, the ReadPort routine is written specifically for the DDT-51 project. Modify the code to suit your application.

```

DOSSeg
.model small
.stack 100h
.data

OutputPort equ 378h
InputPort  equ 379h
ControlPort equ 37Ah

.code

public ReadPort

ReadPort proc

    push dx
    push cx

    mov dx,ControlPort
    mov ax,29h
    out dx,ax                ; assert RD line low
    mov dx,InputPort
    in  al,dx                ; read the nybble
    and al,0F0               ; zero the lower nybble
    shr al,1                 ; faster to do 4 * 1
                             ; bit shifts
    shr al,1
    shr al,1
    mov ch,al                ; save nybble

    mov dx,ControlPort
    mov ax,08h
    out dx,ax                ; pull read line high
    mov dx,InputPort
    in  al,dx                ; read upper nybble
    and al,0F0h              ; zero lower nybble
    or  al,ch                ; merge nybbles into bytes

    pop cx
    pop dx
    ret
endp
end
    
```